COS 597D:
Principles of
Database and Information Systems

# Crash Recovery



## Crash Recovery Overview

- Goals of crash recovery
  - Either transaction commits and is correct or aborts
  - Commit means all actions of transaction have been executed
- Error model:
  - lose contents main memory
  - disk contents intact and correct



## Crash recovery requirements

- If transaction has committed then still have results (on disk)
- If transaction in process, either
  1. Transaction completely aborts

  OR

  2. Transaction can continue after restore as if no crash
- Get serializable schedule such that transactions that committed before crash still commit and in same order

=> NEED LOG



## Tradeoffs?



## ARIES algorithm

- Assumptions
  - Strict 2PL => no cascaded aborts
  - "in place" disk updates: data overwritten on disk
    - Page read into buffer, changed in buffer, written out again
    - Write of page to disk is atomic
- Log:
  - Sequential writes on separate disk
  - Write differences only
    - Multiple updates on single log page
    - Each log record has unique Log Sequence Number
      - LSN strictly sequential



## Log records

- for each transaction, create linked list of log records going back in time
- contents of one log record
  - transaction ID
  - Type
    - update, commit, abort, end, CLR (compensation log record)
  - Update information
    - page ID
    - length & offset
    - before data & after data
  - prevLSN for transaction: link to previous log record

## Bookkeeping: tables

- Transaction table
  - transaction ID
  - status: running, committed, aborted
  - lastLSN
    - points to most recent prevLSN
    - start of chain
- Dirty page table
  - ID of each page with changes not yet on disk
  - recLSN for each page:
    LSN of log record for earliest page change not on disk

These tables in main memory



## Other bookkeeping

**Let buffer manager decide data page writes, so:**

- pageLSN for each data page
  - is LSN of most recent log record for update to that page
  - is stored on data page
- flushedLSN
  - maximum LSN already written to disk
  - is stored in memory
- Requirement: write data page to disk only after write log entries to disk
  - pageLSN <= flushedLSN on data page write



## Checkpoint

- Properties
  - Goes on while other transactions running
    - as separate transaction
  - does not flush dirty pages to disk
  - does tell us how much to fix on crash
- Actions
  1. Write “begin checkpoint” to log
  2. Write current transaction table and dirty page table and “end” as one record to log
     - tables as of “begin checkpoint”
  3. Write log to disk
  4. Store LSN of “begin checkpoint” in safe place
     - “master record”



## Commit

Actions

1. write “commit” to log
2. write to disk all log records up to commit record
3. clean up transaction table,etc.
4. write “end(commit)” record to log

- commit is executed as soon as disk write finishes
- if crash before table clean-up, transaction will commit on recovery



## Update

Actions

1. *Pin* data page in buffer and write change
2. Write log entry (LSN=# )
3. Update transaction table (lastLSN = #)
4. Update dirty page table
5. Write pageLSN= # to page and *unpin* page



## Transactions do concurrently (mixed)

- Commit
- Abort (those not part of restart after crash)
- Checkpoint
- Update

### Crash recovery manager does alone:

All actions during restore of database during restart after crash

## When write to disk?

- Write log pages from buffer:
  - on checkpoint
  - on commit of transaction
  - When want to write data page but pageLSN > flushedLSN
- Write data pages from buffer:
  - At discretion of buffer manager
- Writing fewer log pages and sequentially: cheaper

13

## Crash

- only know about uncommited transactions at last checkpoint - same for dirty pages
- only know about commited transactions since chkpt
- could be changed pages on disk from transactions no longer know about
- lost state of running transactions as processes – can't pick up where left off
- lost knowledge of external effects

14

## Crash recovery Phase I: Analysis

- Get log from disk
- Get most recently checkpointed transaction table and dirty page table
  - use *master record*
- Read log forward from checkpoint and update tables
  - For END log entries, remove transaction from transaction table
  - For other log entries, add or update transaction table entry
    - extends chain of prevLSNs

15

## Crash recovery Phase II: Redo

- REDO all actions in log starting at earliest point when a change not on disk
  - Want earliest recLSN of all recLSNs in dirty pg table
  - Includes redo of UNDOs and ABORTs
    - See Phase III
- When redo action
  - Write new pageLSN
  - Do NOT write new Log entry

16

## At end phase II Redo

- DB now in state was as recorded by *log on disk* at crash
- To finish phase II
  - write END log records for transactions in transaction table that were committed
  - Remove committed transactions from transaction table

17

## Crash recovery Phase III: Undo

- UNDO actions of all transactions not committed by the end of phase II
- Work backwards through log
  - Follow pointer chain from each still-active transaction

    lastLSN →prevLSN →prevLSN → … → prevLSN
  - To process, interleave chains in LSN order from all active transactions
    - Event queue

18

## Phase III UNDO Actions

- For UPDATE
  1. Write CLR record to log *NEW*
     - Records change done to undo UPDATE
     - Records undoNextLSN storing prevLSN of this UPDATE
       - Records next record to undo
     - Think of as ABORT log record like UPDATE log record
  2. Undo change in UPDATE
  3. If prevLSN for UPDATE == NULL, write END record for transaction

     Else queue prevLSN for processing

UNDO makes new DB changes =>
Need step 1 to deal with another crash as undoing



## Phase III UNDO Actions

- For CLR

  If undoNextLSN == NULL, write END record for transaction
  - Undo/abort of transaction done

  Else queue undoNextLSN for processing
  - Re-establishes prevLSN chain for undoing/ aborting transaction

- If are undoing a CLR, were in the process of undoing/ aborting a transaction when crashed
- The redo of the CLR in phase II did the actual undoing
- Don't undo the UNDO represented by CLR record!



## Effects of recovery

- REDO does "clean-up"
  - ends committed transactions
  - Writes ENDs to log
- UNDO does *new work* to undo/abort
  - Changes data pages, which may be on disk
  - Writes log entries for its actions



## Short Example UNDO phase



- T1 and T2 to undo
- Assume neither an ABORT

Queued to undo:
LSN 7
LSN 5



## Short Example UNDO phase



1) Write "CLR LSN7 of T1" to log with undoNextLSN = 3
2) Undo action of T1 in LSN 7
3) Queue prevLSN(7) to undo

Queued to undo:
LSN 5
LSN 3



## Short Example UNDO phase



1) Write "CLR LSN5 of T2" to log with undoNextLSN = null
2) Undo action of T2 in LSN 5

**CRASH**

Queued to undo:
LSN 3

## Short Example UNDO phase

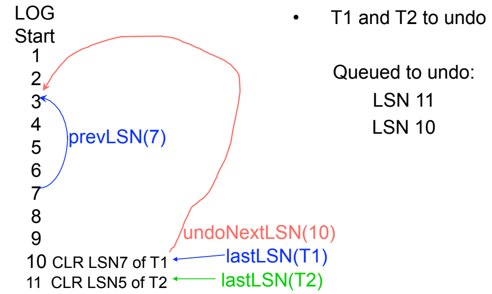
# CRASH

Assume log through entry 11 was written to disk

Look at new UNDO phase

25

## Short Example UNDO phase

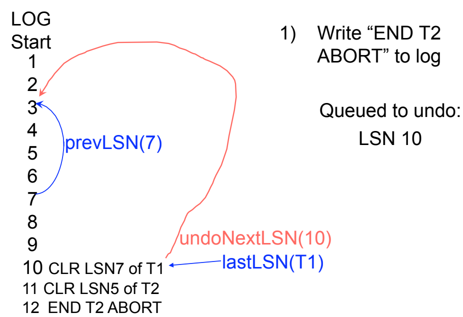



- T1 and T2 to undo

Queued to undo:
LSN 11
LSN 10

26

## Short Example UNDO phase

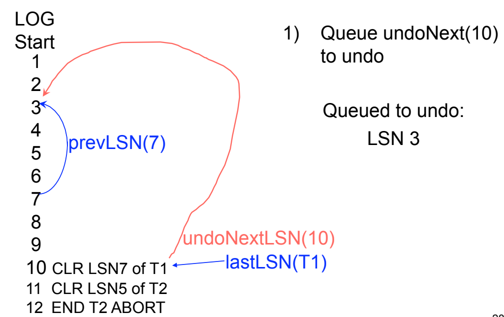



1) Write “END T2 ABORT” to log

Queued to undo:
LSN 10

27

## Short Example UNDO phase

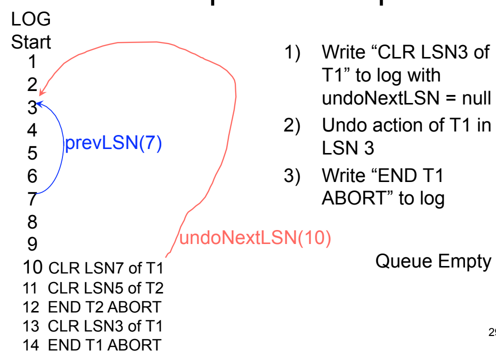



1) Queue undoNext(10) to undo

Queued to undo:
LSN 3

28

## Short Example UNDO phase




1) Write “CLR LSN3 of T1” to log with undoNextLSN = null
2) Undo action of T1 in LSN 3
3) Write “END T1 ABORT” to log

Queue Empty

29

## Abort as part of a transaction

- Write ABORT log record
  - Analogous to COMMIT but more to do before END
- Execute UNDO phase for
  lastLSN →prevLSN →prevLSN → … → prevLSN
  of the aborting transaction
- When UNDO phase writes END to log, is end of ABORT of transaction
  - Must remove from transaction table

30

## Summary

- Choose durability method based on application and balance of costs
  - transaction complexity
  - read time
  - recover time
  - write time
  - space
- Aries goals
  - multi-update RDB transactions
  - minimize disk write
    - data writes by buffer manager as necessary
    - log writes sequential and as necessary
  - minimize read time
    - in-place writes => single look-up
  - reduce recovery time with checkpoints

31